Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications

Valerie Taylor
Department of Computer Science
Texas A&M University
College Station, TX 77843
Phone: 979-8455820
taylor@cs.tamu.edu

Xingfu Wu
Department of ECE
Northwestern University
Evanston, IL 60208
Phone: 847-4917378
wuxf@ece.northwestern.edu

Rick Stevens
MCS Division
Argonne National Laboratory
Argonne, IL 60439
Phone: 630-2523378
stevens@mcs.anl.gov

ABSTRACT
Performance is an important issue with any application, especially grid applications. Efficient execution of applications requires insight into how the system features impact the performance of the applications. This insight generally results from significant experimental analysis and possibly the development of performance models. This paper present the Prophesy system, for which the novel component is the model development. In particular, this paper discusses the use of our *coupling* parameter (i.e., a metric that attempts to quantify the interaction between kernels that compose an application) to develop application models. We discuss how this modeling technique can be used in the analysis of grid applications.

General Terms
Measurement, Performance

Keywords
Performance analysis, performance modeling, grid applications, parallel applications, and grid systems.

1. Introduction
Currently, distributed systems, especially grid systems, are becoming available through programs such as the TeraGrid [TG], the NASA Information Power Grid [JG99], the Alliance [AL], the National Partnership for Advanced Computational Infrastructure [NP], GriPhyN [GP], and the Europen Grid Effort [EG]. Grids, in contrast to conventional parallel systems, have some unique features that pose significant challenges in terms of performance modeling and analysis. These unique features include:

- The resources are heterogeneous.
- The resources at the geographically different sites have different clocks.
- The number of resources is assumed to be a significant number.
- Many of the resources are shared (e.g., the WANs used to interconnect the resources), resulting in dynamic loads.

Performance is an important issue with any application, especially grid applications. Efficient execution of applications requires insight into how the system features impact the performance of the applications. This insight generally results from significant experimental analysis and possibly the development of performance models. This paper presents Prophesy, a web-based performance analysis and modeling infrastructure for parallel and grid applications. Prophesy includes an extensive database that archives the details about the context in which the performance data was collected as well as the performance data itself. In terms of the clocks, the Prophesy instrumentation tool collects aggregate information about sections of an application code; hence log files involving timestamps are not used. Further, performance information is collected per processor and automatically sent to the database for archiving at the end of execution of an application. The archival of the performance data is independent of the application execution, so as not to hinder or impede the execution.

The novel aspect about Prophesy is the automated modeling component that includes conventional as well as a new techniques for developing performance models. This paper focuses on this modeling component, detailing how the techniques support grid applications. In particular, the paper discusses the use of our *coupling* parameter, a metric that attempts to quantify the interaction between kernels that compose an application. The metric is used to identify how to combine the performance models of the kernels that compose an application into a model of the application. We discuss how this technique can be used with grid applications. Our future work is focused on how to incorporate the dynamic load information into the performance models to get better predictions.

The remainder of this paper is organized as follows. Section 2 presents the background information about Prophesy followed by a detailed presentation of the modeling component in Section 3. Related work is given in Section 4, followed by the paper summary in Section 5.

2. Prophesy Framework

The Prophesy framework consists of three major components: data collection, data analysis, and three central databases, as illustrated in Figure 1. The data collection component focuses on the automatic instrumentation of codes at the level of basic blocks, procedures, or loops. The default mode consists of instrumenting the entire code at the level of loops and procedures. A user can specify that the code be instrumented at different levels of granularity or manually insert directives for the instrumenting tool to instrument specific segments of code. The resultant performance data is automatically placed in the performance database. This data is used by the data analysis component to produce an analytical performance model at the level of granularity specified by the user, or answer queries about the best implementation of a given function. The models are developed based upon performance data from the performance database, model templates from the template database, and system characteristics from the systems database. These models can be used to predict the performance of the application under different system configurations. The Prophesy interface uses web technology to enable users from anywhere to access the performance data, add performance data, or utilize the automated instrumentation and modeling processes.

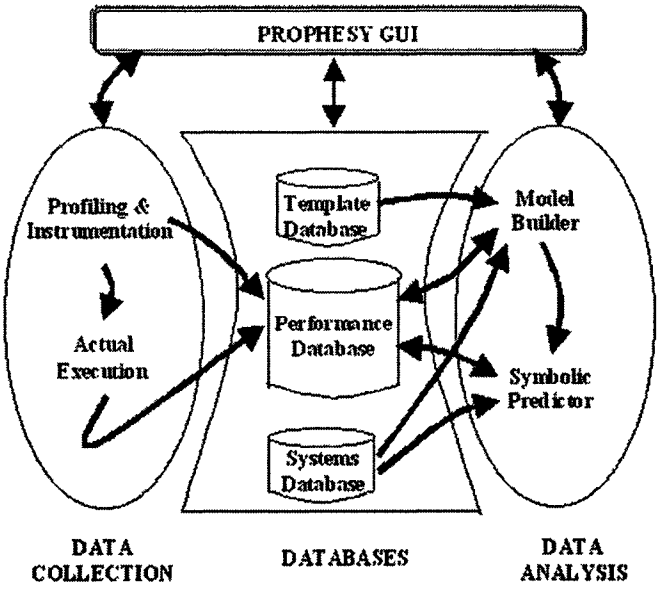

Figure 1. Prophesy framework.

2.1 Data Collections

PAIDE (Prophesy Automatic Instrumentation and Data Entry), shown in Figure 2, is the data collection component of the Prophesy system; the goal of PAIDE is to minimize instrumentation overhead [WT01b]. PAIDE includes a parser that identifies where to insert instrumentation code. PAIDE also generates two files: (1) the call graph of the application and (2) the locations in the code where instrumentation was inserted. The information in these two files allow the performance data to be directly related to the application code for code tuning. For each execution of an application, PAIDE records the time of day and the IP address of node 0 of the grid system used to execute the application; this information forms a unique identifier to allow data from different files to be associated with the same execution.

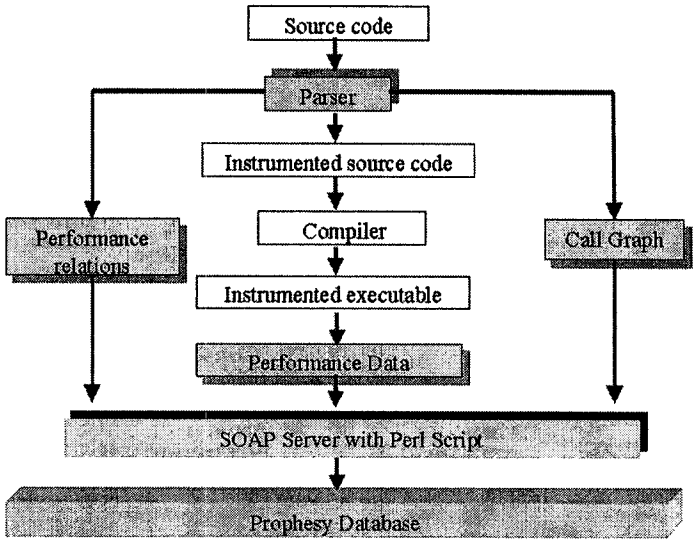

Figure 2. PAIDE framework.

2.2 Prophesy Database

Prophesy assumes that applications can be decomposed into modules, which can be further decomposed into functions that can be decomposed into basic units in a hierarchical manner as depicted in Figure 3. In particular, we assume the following meaning about each component:

- **Application**: refers to the complete large-scale application.
- **Modules**: refer to the various files that comprise the application; it is assumed that the application designer uses some modularity in the application design.
- **Functions**: refer to the different function routines that may be contained in a given module. Users will be asked to associate a "pure function" name with their given function where appropriate. For example, a user may identify their function "genfft" as the pure function FFT. Pure functions are widely used functions such as conjugate gradient or Gaussian elimination. Pure functions facilitate the best implementation queries.
- **Basic Units**: refer to a code segment that may be of finer granularity than a function but coarser granularity than a basic block. For example, a segment of nested loops would be considered one basic unit.

The Prophesy database shown in Figure 4 also has a hierarchical organization, consistent with the hierarchical structure of the applications. The schema given below includes all three databases given in Figure 1: performance database, system models database and template database. The entities in the database are organized into four areas: application information,

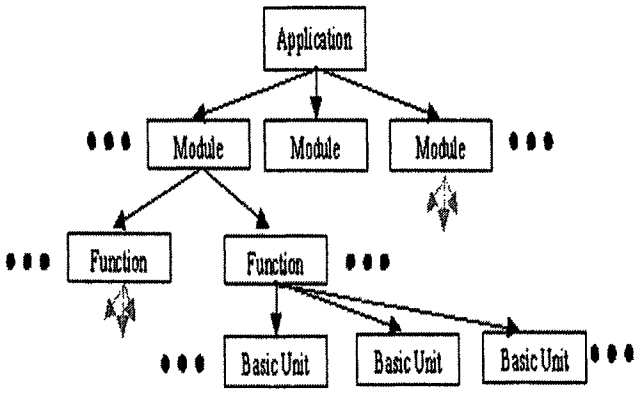

Figure 3. Hierarchical structure of an application

executable information, run information and performance statistics, as described below:

- **Application Information**: includes one entity that gives the application name, version number, a short description, and owner information and password (such that only the owner can modify or add data for a given application). Data is placed into this entity when a new application is being developed.
- **Executable Information**: includes all of the entities related to generating an executable of an application. These entities include details about compilers, libraries (compile time and run time) and the control flow. Data is placed into these entities when a new executable is being used.
- **Run Information:** includes all of the entities related to running an executable, which includes the system information and inputs used for execution. *This system may be a single processor, single parallel machine or grid system.* Data is placed into these entities for each run of a given executable. Further, detailed information about different systems (e.g., processor performance, node memory subsystem, operating system, etc.) is contained in this area.
- **Performance Statistics Information**: includes all of the entities related to the raw performance data collected during execution. Performance statistics are collected for all levels of the application hierarchy.

2.3 Automated Modeling

Prophesy's automated modeling component allows models to be developed easily and stores relevant modeling information in the database for later use. This latter feature results in a modeling process that improves over time. Currently, Prophesy's framework supports the following modeling techniques: curve fitting, parameterization, and kernel coupling methods. The first two methods are well-established techniques, for which Prophesy facilitates the methods via automation. The last method, kernel coupling, is enabled by Prophesy because of the significant amount of archival data and the automation of the modeling process. In this work, a kernel is a unit of computation that denotes a logical entity within the larger context of an

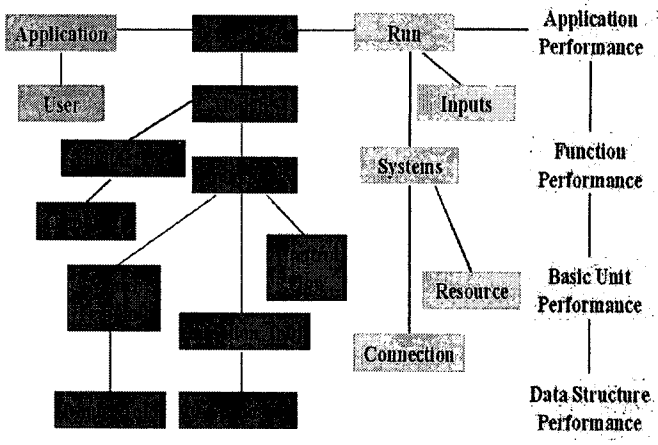

Figure 4. PROPHESY Database Schema

application. The unit may be a loop, procedure, or file depending on the level of granularity of detail that is desired from the measurements. This method is to use kernel models to develop full application models. In this section we describe, briefly, what is done for curve fitting and parameterization; the coupling method is described in Section 4, as this method relates to grid applications.

2.3.1 Curve Fitting

Curve Fitting is a method that uses optimization techniques, e.g., least squares, to develop a model. For this method, Prophesy uses the empirical data found in the database. The empirical data to be used for the fit is determined by the user. Then GNU Octave [OC] is used to generate the resultant model. The advantage of this method is the ease for which the analytical model is generated; the disadvantage is the lack of exposure of system terms versus application terms. The models resulting from curve fitting are generally a function of some input parameters of the application; the system performance (e.g., operation execution time or communication performance) is lumped into the coefficients determined by curve fitting. Hence, models resulting from curve fitting can be used to explore application scalability but not different system configurations.

2.3.2 Parameterization Method

Parameterization is a method that combines manual analysis of the code with system performance measurements. The manual analysis entails hand-counting the number of different operations in the code. It is assumed that this type of analysis is done on kernels or functions that are generally in the range of 100 lines of code or less. With Prophesy, the manual analysis is used to produce an analytical equation with terms grouped together such that the equation is a function of some input variables. For example, given a matrix-vector multiply kernel, the complexity is quadratic in terms of the matrix rank. The manual analysis would entail grouping the terms to produce something such as $\alpha_1 N^2 + \alpha_2 N + \alpha_3$, whereby α_1, α_2 and α_3 have explicit terms representing the application and the system and N, the matrix rank. These terms are represented in the Prophesy database as scripts that can be used to generate the values based upon data in the database. Having the system and application terms represented explicitly, one can use the resultant models to explore what happens under

different system configurations as well as application sizes. The disadvantage of this method is the time required for manual analysis. However, given that the focus is on functions or kernels for manual analysis, the time requirement is feasible. Further, the manual analysis is done only once per function or kernel code. Lastly, we believe that the set of kernels will remain relatively small as indicated by previous work identifying the large number of applications affected by research in a fixed number of areas.

3. Kernel Coupling

Kernel coupling refers to the effect that *kernel i* has on *kernel j* in relation to running each kernel in isolation. The two kernels can correspond to adjacent kernels in the control flow of the application or a chain of three or more kernels. In our previous work we used coupling to identify parts of the application that required performance improvement [GT99]. The coupling value provided insight into where further algorithm and code implementation work was needed to improve performance, in particular the reuse of data between kernels. Current work is focused on demonstrating how the coupling values of adjacent pairs and chains of kernels can be used to develop analytical models for one code [TW01, TW02]. In this section, we first describe how coupling values are generated and then demonstrate how it is used to develop models of applications. We also provide data illustrating that coupling values can be reused.

3.1 Coupling Parameter

The kernel coupling parameter, C_{ij}, quantifies the interaction between adjacent kernels in an application. To compute the parameter C_{ij}, three measurements must be taken:

- P_i is the performance of *kernel i* alone,
- P_j is the performance of *kernel j* alone, and
- P_{ij} is the performance of *kernels i* and *j* (assuming *kernel i* immediately precedes *kernel j*) in the application.

These measurements are done in the sequence determined by the application. In particular, a measurement is obtained by placing a given kernel or pair of kernels into a loop, such that the loop dominates the execution time. Then the time required for the application, beyond the given kernel or pair of kernels, is subtracted such that the resultant time reflects that of only the given kernel or pair of kernels.

We define that, the value C_{ij} is equal to the ratio of the measured performance of the pair of kernels to the expected performance resulting from combining the isolated performance of each kernel. Since C_{ij} is the measurement of interaction between kernels, we compute it as the ratio of the actual performance of the kernels together to that of no interaction as given below:

$$C_{ij} = \frac{P_{ij}}{P_i + P_j} \quad (1)$$

Now, let's consider Equation 1 for the case of an ordered chain of kernels. Let W be the set of all kernels in the ordered chain of k kernels.. Assume that C_W is the coupling value of the chain, and P_W is the performance of the chain. Then, the above equation is modified into

$$C_W = \frac{P_W}{\sum_{l \in W} P_l} \quad (2)$$

Notice that interactions between all pairs or chains of kernels are *not* necessary. The value C_W from Equation 2 represents the direct interaction between two adjacent or chain of kernels in an application (i.e., in the sense of the control flow of the application). We group the kernel couplings into three sets:

- $C_W = 1$: indicates no interaction between the kernels, yielding no change in performance.
- $C_W < 1$: results from some resource(s) being shared between the kernels, producing a performance gain (i.e., constructive coupling).
- $C_W > 1$: occurs when the kernels interfere with each other, resulting in a performance loss (i.e., destructive coupling).

3.2 Application Modeling

Assume that an application has four kernels (A, B, C, D) that are executed together in one loop. Let E_A, E_B, E_C and E_D represent the analytical models of each of the respective kernels; these models include the number of times the kernel is executed in the application. The equation for the estimated application execution time is given as follows:

$$T = \alpha E_A + \beta E_B + \gamma E_C + \delta E_D$$

Using the pair-wise performance coupling values as given in Equation 1, the coefficients have the following values, corresponding to the weighted average of the coupling values containing each kernel:

$$\alpha = [(C_{AB} * P_{AB}) + (C_{DA} * P_{DA})] / (P_{AB} + P_{DA})$$
$$\beta = [(C_{AB} * P_{AB}) + (C_{BC} * P_{BC})] / (P_{AB} + P_{BC})$$
$$\gamma = [(C_{BC} * P_{BC}) + (C_{CD} * P_{CD})] / (P_{BC} + P_{CD})$$
$$\delta = [(C_{CD} * P_{CD}) + (C_{DA} * P_{DA})] / (P_{CD} + P_{DA})$$

The above is denoted as the pair-wise kernel coupling predictor. We could also use three-kernel coupling values for which we would use the coupling values, such as C_{ABC} and the performance P_{ABC}.

In [TW02], we demonstrated the advantages of using the coupling values to estimate performance using the Nas Parallel Benchmarks [BH95]. For BT (Block Tridiagonal) dataset A, the four kernel predictor had an average relative error of 0.79%, while merely summing the times of the individual kernels resulting in an average relative errof of 21.80%. For the SP dataset A, the four kernel predictor had an average relative error of 14.16%, while the summation methodology had an average relative errof of 35.43%.

One of the issues related to the coupling predictor was how many experiments are needed to get good coupling values. In particular, does one need coupling values for each dataset, each system, and different number of processors. Our work in this area has demonstrated that similar systems, such as distributed memory

versus shared memory, have very similar coupling values. Further, with different number of processors, the coupling values have very few distinct changes. In particular, significant changes occur when the is a distinct change in the memory footprint of the application as it scales.

3.3 Relationship to Grid Applications

As discussed previously, grid applications have some unique characteristcs as the execution environment consists of resources at geographically different sites. Prophesy can be used to model grid applications by using a combination of the paramterization method with the coupling method. This can be done in the following manner. First we start off with an application decomposed into a finite set of kernels (e.g., FFT, matrix-matrix multiply, solver, etc.) ; it is assumed that this number is small. Second, we use the parameterization modeling technique to develop performance models of the kernels for each of the different systems used in the execution environment. In particular, the parameterized technique would untilize the different timings for compute and system operations stored in the system database. Further, the parameterized technique would also utilize the timings about the interconnection between systems as well. Then, for each system, we develope a model that utilizes the coupling values for the given system type as well as the granularity of the dataset size. In particular, the coupling values would identify how to combine the kernel models for each system. The resultant models would provide insight into the overall system performance as well as the individual system performance. Currently, we are applying this technique to a grid cosmological application.

4. RELATED WORK

There exist some approaches to organizing performance data by using database techniques. For example, Snodgrass [SN88] has developed a relational approach to monitoring complex systems by storing the information processed by a monitor into a historical database. The basic idea is to use historical databases to formalize dynamic information. The SIEVE (Spreadsheet based Interactive Event Visualization Environment) system [SG93] maintains dependence graph information in a static data base and tracefile information in a dynamic data base. Users may select columns from spreadsheet and associate those with graphical objects for display. The PDS (Performance Database Server) system [HB94] was specifically designed with a simple tabular format that involves displaying the data in rows (machine configuration) and columns (numbers). It logically organizes data according to the benchmarks themselves. Further, it only provides reference performance of a benchmark on various machines.

Significant work has been done with developing performance tools such as Pablo [RA93], AIMS [YS95], or Paradyn [MC95]., and performance analysis environments, in particular PACE [KH96] and POEMS [PO98].. These environments focus on performance predication in contrast to Prophesy for which the focus is on archival of data and model development.

5. Summary

Grid applications are emerging as applications are focusing on very complex and large-scale problems. Performance of this class of applications is important, for which the underlying systems present some unique characteristics. In this paper, we presented Prophesy, a framework for analyzing and modeling the performance of parallel and grid applications. The novel aspect about Prophesy is the automated modeling component, which includes a method for developing models as composition of the performance models of the kernels that compose the application. The combination of using the coupling parameters with parameterized models that are system dependent, allows one to better understand what is going on within each system as well as across systems.

6. ACKNOWLEDGMENTS

This work is supported in part by NSF NGS grant EIA-9974960, NASA Ames, and NSF ITR grants---ITR-0086044 (GriPhyN) , ITR-0085952 , and ITR-0225642 (OptiPuter).

7. REFERENCES

[AL] The Alliance, http://www.alliance.uiuc.edu/

[BH95] D. Bailey, T. Harris, et al., *The NAS Parallel Benchmarks*, Tech. Report NAS-95-020, Dec. 1995. See also http://science.nas. nasa.gov/Software/NPB/.

[TG] TeraGrid Project, http://www.teragrid.org/

[GP] GriPhN Project, http://www.griphn.org/

[TW01] V. Taylor, X. Wu, J. Geisler, X. Li, Z. Lan, M. Hereld, I. Judson, R. Stevens, Prophesy: Automating the Modeling Process, *Invited Paper*, in *Proc. of the 3rd International Workshop on Active Middleware Services 2001*, in conjunction with HPDC-10, August 2001.

[TW02] V. Taylor, X. Wu, J. Geisler, and R. Stevens, Using Kernel Couplings to Predict Parallel Application Performance, in *Proc of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC2002)*, Edinburgh, Scotland, July 24-26, 2002.

[GT99] J. Geisler and V. Taylor, Performance coupling: A methodology for predicting application performance using kernel performance, in *Proc. of the 9th SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.

[SN88] R. Snodgrass, A relational approach to monitoring complex systems, *ACM Transactions on Computer Systems*, Vol. 6, No. 2, May 1988, 157-196.

[SG93] S. R. Sarukkai, and D. Gannon, SIEVE: A performance debugging environment for parallel program, *Journal of Parallel and Distributed Computing* 18, 1993, 147-168.

[HB94] R. Hockney and M. Berry, *Public International Benchmarks for Parallel Computers*, PARKBENCH Committee: Report-1, February 7, 1994.

[RA93] D. A. Reed, R. A. Aydt, et al., Scalable performance analysis: The Pablo performance analysis environment. In *Proc. of the Scalable Parallel Libraries Conference*, Oct. 1993.

[YS95] J. C. Yan, S. R. Sarukkai, and P. Mehra, Performance measurement, visualization and modeling of parallel and

distributed programs using the AIMS toolkit, *Software Practice and Experience*, Vol. 25 (4), April 1995.

[MC95] B. P. Miller, M. D. Callaghan, et al., The Paradyn parallel performance measurement tools, *IEEE Computer*, Vol. 28 (11), Nov. 1995.

[KH96] D. Kerbyson, J. Harper, et al., PACE: A toolset to investigate and predict performance in parallel systems. In *Proc. of European Parallel Tools Meeting*, Oct. 1996.

[OC] GNU Octave Homepage, http://www.octave.org.

[PO98] POEMS Performance Environment, http://www.cs.utexas.edu/users/poems

[NP] The National Partnership for Advanced Computational Infrastructure, http://www.npaci.edu/

[EG] The European Grid, http://www.ggf1.nl/.

[FK98] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, July 1998.